Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій

УДК 004.9

На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: "Проектування та розробка клієнтської частини веб застосування для ведення бази даних публікацій"

Спеціальність – 121 "Інженерія програмного забезпечення"

ПОЯСНЮВАЛЬНА ЗАПИСКА

Студент
ІПЗ-41_____ /Максим СТОЛЯР/
Науковий керівник
к.ф.м.н.,доц._____/ Сергій ПОЛЯКОВ/
Консультант
з питань нормоконтролю
фахівець_____/ Тамара ЧАПОВСЬКА/

Допускається до захисту
Завідувач кафедри
д.т.н.,доц._____/Олексій БИЧКОВ/

Київ – 2021

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

Освітньо-кваліфікаційний рівень бакалавр

Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖЕНО

Зав. кафедри програмних систем і технологій

_____(Олексій БИЧКОВ)

(підпис) (прізвище та ініціали)

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Столяру Максиму Олександровичу

---

1. Тема бакалаврської роботи “ Проектування та розробка клієнтської частини веб застосування для ведення бази даних публікацій”, керівник проекту (роботи) Поляков Сергій Анатолійович, к.ф.-м.н., доцент_____ затверджені наказом вищого навчального закладу від “ ___ ”2021 р. № _____

2. Строк подання студентом роботи _____ 2021 р.

3. Вихідні дані до проекту (роботи) Базові концепції та розуміння проектування та розробки програмних технологій

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Вибір середи розробки та її основні переваги

2. Проектування програмного забезпечення та допоміжні засоби для цього

3. Використання сторонніх бібліотек

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

1. Представлення DOM (рис. 1.1, ст. 14)

2. Представлення AJAX (рис. 1.2, ст. 15)

3. Перехід по сторінкам 1 (рис. 1.3, ст. 16)

4. Перехід по сторінкам 2 (рис. 1.4, ст. 17)

5. MVVM (рис. 1.5, ст. 19)

6. Показ помилки (рис. 2.1, ст. 31).

7. Вивід усіх даних (рис. 2.2, ст. 32)

8. Вивід потрібних даних (рис. 2.3, ст. 33)

9. Форма реєстрації (рис. 3.1, ст. 39)

10. Вхід у систему (рис. 3.2, ст. 44)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Проектування бази даних	Поляков Сергій Анатолійович		

7. Дата видачі завдання _____ 2021 р.

Керівник _____ (Сергій ПОЛЯКОВ)

Завдання прийняв до виконання _____(Максим СТОЛЯР)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів бакалаврської роботи	Термін виконання етапів роботи	Примітка
1	Підбір та вивчення літератури	10.01.2021	Виконано
2	Аналіз можливих алгоритмів	15.01.2021	Виконано
3	Вивчення додаткових бібліотек	13.02.2021	Виконано
4	Розробка алгоритмічної моделі	17.02.2021	Виконано
5	Опис розробленого алгоритму	25.03.2021	Виконано
6	Затвердження пояснювальної записки роботи завідуючого кафедри	17.04.2021	Виконано

Студент – бакалавр _____(Максим СТОЛЯР)

Керівник роботи _____(Сергій ПОЛЯКОВ)

АНОТАЦІЯ

Випускна кваліфікаційна бакалаврська робота: 68 с., 10 рис., 2 додат.,4 джерела.

Тема: Проектування та розробка клієнтської частини веб застосування для ведення бази даних публікацій

Об'єкт дослідження: публікації різних авторів та додаткові закріплені дані до публікації.

Мета роботи: розробка клієнтської частини для публікацій для різних авторів.

Предмет дослідження: технологія клієнтської частини та обробки різних видів запитів на сервер з подальшою відповіддю.

Результати дослідження: Досліджено можливості застосування клієнтської частини веб застосування для повноцінної роботи з серверною частиною. Запропонований логічний підхід для вирішення цієї задачі.

Висновок

В результаті досліджень була розроблена та протестована повністю функціонуюча програма, яка здатна обробляти вхідні та вихідні дані з серверу та відсилати запити до бази даних.

БАЗА ДАНИХ, ПРОГРАМА, КЛІЄНТ, РОЗРОБКА, СЕРВЕР, ЗАПИТ, ПРОГРАМУВАННЯ.

SUMMARY

Final qualifying bachelor's thesis:68 pages,10 pictures,2 appendices,4 sources.

Topic: Design and development of the client part of the web application for maintaining a database of publications

Object of research: publications of various authors and additional attached data to the publication.

Purpose: development of the client part for publications for different authors.

Subject of research: client technology and processing of various types of requests to the server with subsequent response.

Research results: Possibilities of application of the client part of the web application for full-fledged work with the server part are investigated. A logical approach to solve this problem is proposed.

Conclusion

As a result of research, a fully functional program was developed and tested, which is able to process input and output data from the server and send requests to the database.

DATABASE, PROGRAM, CLIENT, DEVELOPMENT, SERVER, REQUEST, PROGRAMMING.

АННОТАЦИЯ

Выпускная квалификационная бакалаврская работа: 68 с., 10 рис., 2 доп., 4 источника.

Тема: Проектирование и разработка клиентской части веб приложение для ведения базы данных публикаций

Объект исследования: публикации различных авторов и дополнительные закреплены данные к публикации.

Цель работы: разработка клиентской части для публикаций для различных авторов.

Предмет исследования: технология клиентской части и обработки различных видов запросов на сервер с последующим ответом.

Результаты исследования: Исследованы возможности применения клиентской части веб приложение для полноценной работы с серверной частью. Предложенный логический подход для решения этой задачи.

Вывод

В результате исследований была разработана и протестирована полностью функционирующая программа, которая способна обрабатывать входящие и исходящие данные с сервера и отправлять запросы к базе данных.

БАЗА ДАННЫХ, ПРОГРАММА, КЛИЕНТ, РАЗРАБОТКИ, СЕРВЕР, ЗАПРОС, ПРОГРАММИРОВАНИЕ.

ЗМІСТ

Стр.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ……..……9

ВСТУП……..10

РОЗДІЛ 1

ПЛАНУВАННЯ ТА ОБРАННЯ ТЕХНОЛОГІЙ ТА БІБЛІОТЕК ДЛЯ РОЗРОБКИ

1.1 Технології для розробки клієнтської частини……..….13

1.2 Проектування та створення додатку за допомогою фреймворку Vue та бібліотек…….....….18

РОЗДІЛ 2

МАРШРУТИЗАЦІЯ. ПІДКЛЮЧЕННЯ ТА НАЛАШТУВАННЯ РОУТЕРІВ ТА ІНШИХ ДОПОМІЖНИХ БІБЛІОТЕК

2.1 Маршрутизація ……..……...22

2.2 Бібліотека axios для взаємодії з REST API …….28

РОЗДІЛ 3

ІНТЕРФЕЙС ПРОЕКТУ ТА ЙОГО ФУНКЦІОНАЛ

3.1 Реєстрація користувача..……..……...34

3.2 Вхід користувача……. …….40

ВИСНОВОК…….……46

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ……..……...47

ДОДАТКИ……48

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД	-	база даних
ПЗ	-	програмне забезпечення
VScode	-	visual studio code
HTML	-	Hypertext markup language
CSS	-	Cascading Style Sheets
JS	-	JavaScript
DOM	-	document object model
JWT	-	JSON Web Token

ВСТУП

Актуальність роботи

Програмне забезпечення такого типу являється одним з найбільш затребуваним на ринку на сьогоднішній час, починаючи від обраних фреймворків та бібліотек, які входять у розробку.

Насамперед було обрано ті стеки технологій, які є актуальними на ринку, завдяки своїм перевагам та зручності.

Порівняння роботи з відомими розв'язаннями проблеми

На даний момент конкретно цього аналогу програмного забезпечення не існує, тому цей веб додаток є оригінальним.

Авжеж ми можемо знайти дещо спільного з відомими соціальними мережами на кшталт facebook або twitter тому, що в деяких моментах вони вирішують схожі проблеми. Але кожне ПЗ є оригінальним і написане під свої вимоги.

В цій роботі були впроваджені основні рішення виникаючих проблем та рекомендації інших інженерів програмного забезпечення з різноманітних сайтів, статей, форумів, книжок та інших джерел інформації. Використання цих технологій та алгоритмів вплинуло на оптимальність та ефективність даного програмного забезпечення.

Мета і задачі дослідження

Метою бакалаврської роботи є проектування та розробка клієнтської частини веб застосування для ведення бази даних публікацій для взаємодії з ПЗ збоку серверної частини та самими користувачами.

Досліджувана модель повинна мати можливість взаємодіяти базою даних та мати можливість виконання таких **задач**:

- Реєстрація користувача
- Вхід користувача у систему
- Додавання даних
- Вивід даних
- Корегування даних

Особливу увагу потрібно звернути на перші два пункта, оскільки вони мають деякі обмеження щодо їх подальшого використання та контролю як на клієнтській так і на серверній частини.

Об’єктом дослідження є публікації різних авторів, представлені у вигляді наукових статей.

Предметом дослідження є технологія клієнтської частини Front-end розробки, яка має змогу приймати та передавати дані через REST API з серверної частини.

Методи дослідження

Для роботи з програмним забезпеченням були використані методи моделювання реляційних баз даних та використання нормальних форм проектування реляційних структур даних.

Практичне значення одержаних результатів

Отримана реалізація розробленого інтерфейсу застосунку та основна логіка робото спроможності програми дає можливість надати користувачеві скористатися даним програмним продуктом.

Особистий внесок студента

Основним результатом є:

1. Власний підхід до розробки інтерфейсу та функціоналу програми.
2. Створення особистих алгоритмів для відображення результатів ПЗ.

Структура та обсяг роботи даних

Робота викладена на 63 сторінках друкованого тексту. Робота містить 10 рисунків та 1 додаток, обсягом 15 стор.

РОЗДІЛ 1

ПЛАНУВАННЯ ТА ОБРАННЯ ТЕХНОЛОГІЙ ТА БІБЛІОТЕК ДЛЯ РОЗРОБКИ

1.1 Технології для розробки клієнтської частини

Front-end веб-розробка - це практика перетворення даних у графічний інтерфейс за допомогою HTML, CSS та JavaScript, щоб користувачі могли переглядати та взаємодіяти з цими даними.

Hypertext markup language

Мова розмітки гіпертексту (HTML) є основою будь-якого процесу розробки веб-сайтів, без якого веб-сторінка не існує. Гіпертекст означає, що в текст вбудовані посилання, які називаються гіперпосиланнями. Коли користувач натискає слово або фразу, що має гіперпосилання, він відкриє іншу веб-сторінку. Мова розмітки вказує, що текст можна перетворити на зображення, таблиці, посилання та інші зображення. Саме HTML-код забезпечує загальну структуру того, як буде виглядати сайт. HTML був розроблений Тімом Бернерсом-Лі. Остання версія HTML називається HTML5 і була опублікована 28 жовтня 2014 року за рекомендацією W3. Ця версія містить нові та ефективні способи обробки таких елементів, як відео та аудіофайли.

Cascading Style Sheets

Каскадні таблиці стилів (CSS) контролюють аспект презентації сайту та дозволяють вашому сайту мати свій унікальний вигляд. Він робить це, підтримуючи таблиці стилів, які сидять поверх інших правил стилів і запускаються на основі інших входів, таких як розмір екрана пристрою та роздільна здатність.

JavaScript

JavaScript - це імперативна мова програмування на основі подій (на відміну від декларативної моделі мови HTML), яка використовується для перетворення статичної HTML-сторінки в динамічний інтерфейс. Код JavaScript може використовувати об'єктну модель документа (DOM), передбачену стандартом HTML, для маніпулювання веб-сторінкою у відповідь на події, такі як введення користувачами.

Об'єктна модель документа (DOM) - це міжплатформенний та незалежний від мови інтерфейс, який обробляє XML або HTML-документ як деревну структуру, де кожен вузол є об'єктом, що представляє частину документа. DOM представляє документ з логічним деревом. Кожна гілка дерева закінчується вузлом, і кожен вузол містить об'єкти. Методи DOM дозволяють отримати програмний доступ до дерева; за допомогою них можна змінити структуру, стиль або зміст документа. На вузлах можуть бути прикріплені обробники подій. Після активації події виконуються обробники подій.

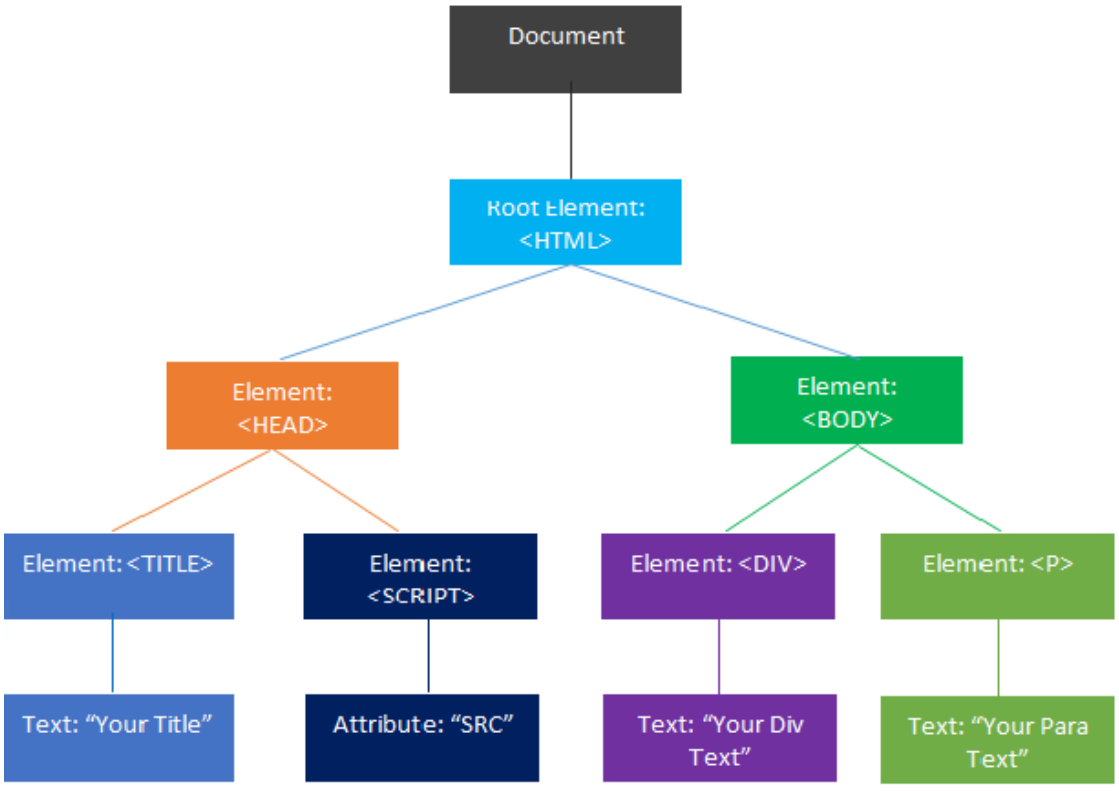


Рис. 1.1 Представлення DOM

Використовуючи на практиці техніку, яка називається AJAX, код JavaScript може також активно отримувати вміст з Інтернету (незалежно від пошуку оригінальної сторінки HTML), а також реагувати на події на стороні сервера, додаючи справді динамічний характер роботи веб-сторінки.

Ajax (також скорочення AJAX - "Асинхронний JavaScript та XML") - це набір веб-технологій розробки, що використовують безліч веб-технологій на стороні клієнта для створення асинхронних веб-додатків. За допомогою Ajax веб-програми можуть надсилати та отримувати дані із сервера асинхронно (у фоновому режимі), не втручаючись у відображення та поведінку існуючої сторінки. Від'єднуючи шар обміну даними від рівня презентації, Ajax дозволяє веб-сторінкам і, за розширенням, веб-програмам, динамічно змінювати вміст без необхідності перезавантажувати всю сторінку. У нашому випадку було використовано JSON замість XML.

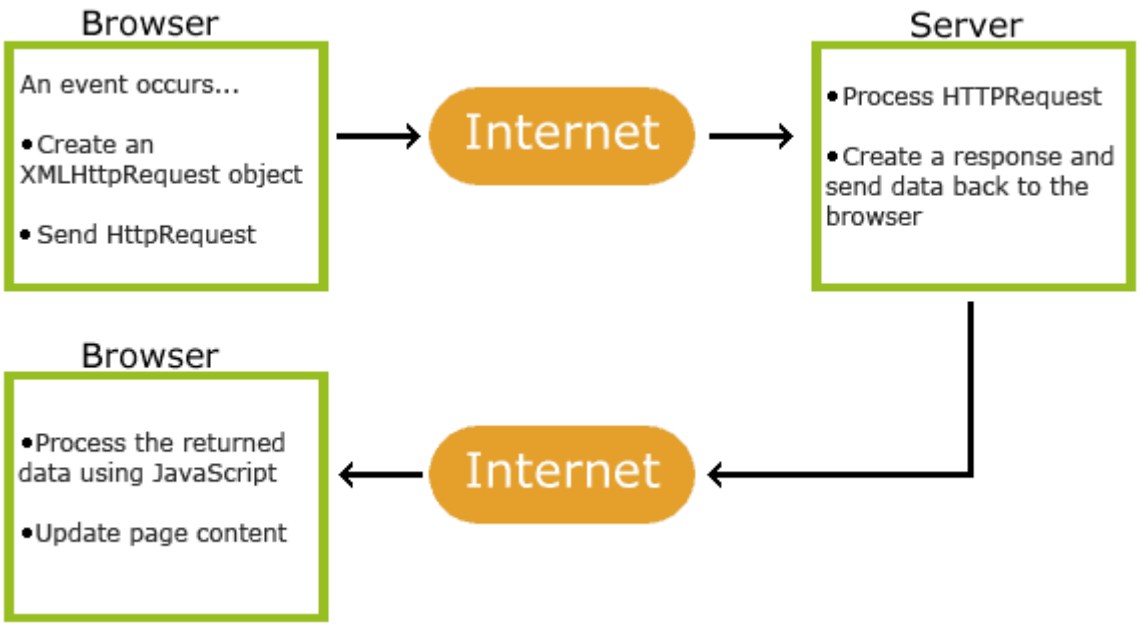


Рис. 1.2 Представлення AJAX

Ajax - це не окрема технологія, а скоріше група технологій. HTML і CSS можна використовувати в поєднанні для розмітки та стилізації інформації. Потім веб-сторінка може бути змінена за допомогою JavaScript, щоб динамічно відображати нову інформацію та дозволяти користувачеві взаємодіяти з нею.

Вбудований об'єкт XMLHttpRequest, або з 2017 року нова функція отримання в JavaScript, зазвичай використовується для запуску Ajax на веб-сторінках, дозволяючи веб-сайтам завантажувати вміст на екран, не оновлюючи сторінку. Ajax - це не нова технологія чи інша мова, а лише існуючі технології, що використовуються по-новому.

Це дало змогу проекту виконувати асинхронні дія без перезавантаження при переході на інші сторінки(Рис.1.3 та Рис. 1.4).

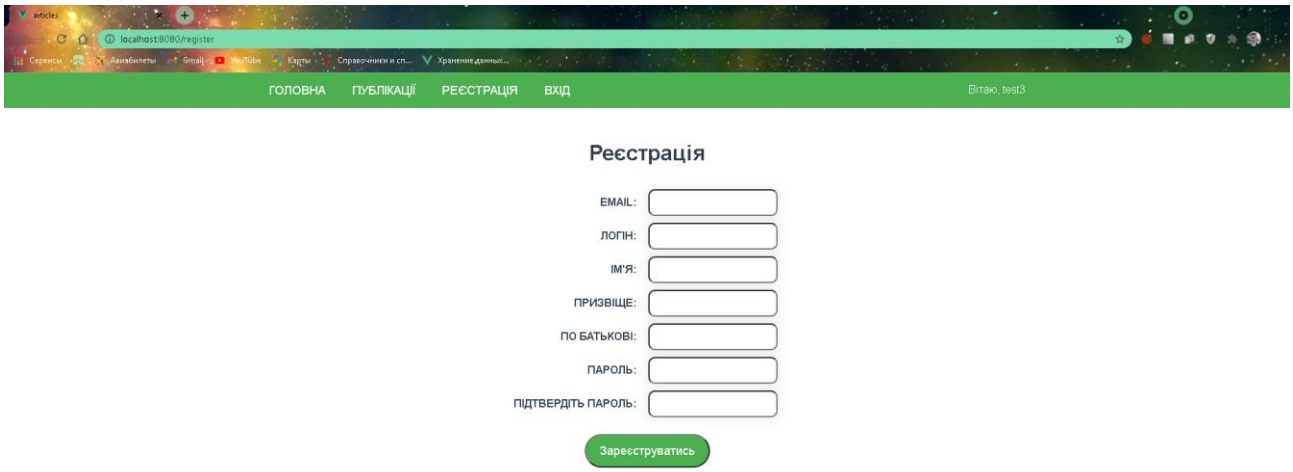


Рис. 1.3 Перехід по сторінкам 1

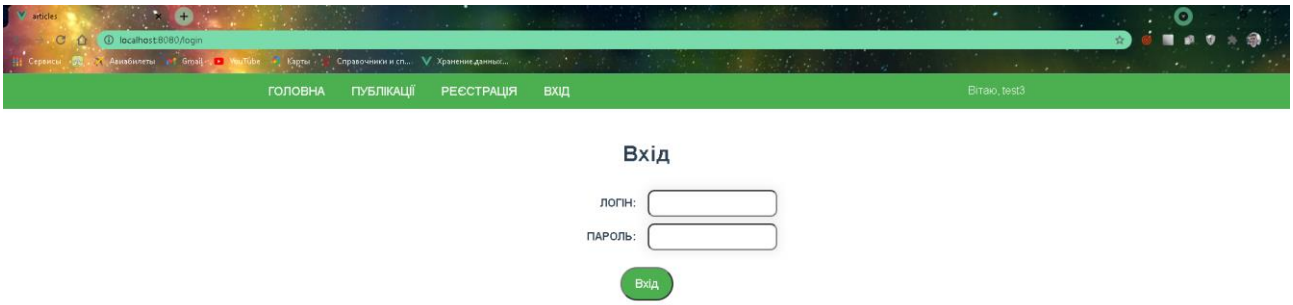


Рис. 1.4 Перехід по сторінкам 2

Даний проект написаний на платформі Visual Studio Code.

Visual Studio Code - це легкий, але потужний редактор вихідного коду, який працює на вашому робочому столі та доступний для Windows, macOS та Linux. Він постачається із вбудованою підтримкою JavaScript, TypeScript та Node.js і має багату екосистему розширень для інших мов (таких як C ++, C #, Java, Python, PHP, Go) та середовищ виконання (таких як .NET та Unity) .

Корпорація Майкрософт випустила більшість вихідних кодів Visual Studio Code у сховищі microsoft / vscode (Code - OSS) GitHub під дозволеною ліцензією MIT, тоді як версії Microsoft є власною безкоштовною програмою.

1.2 Проектування та створення додатку за допомогою фреймворку Vue та бібліотек

У розробці проекту я буду використовувати фреймворк Vue

Vue - це прогресивний фреймворк для побудови користувальницьких інтерфейсів. На відміну від інших монолітних каркасів, Vue розроблений з нуля, щоб бути поступово впровадженим. Його ядро в першу чергу вирішує завдання рівня уявлення (view), що спрощує інтеграцію з іншими бібліотеками та існуючими проектами. З іншого боку, Vue повністю підходить і для створення складних односторінкових додатків (SPA, Single-Page Applications), якщо використовувати його спільно з сучасними інструментами та додатковими бібліотеками.

Vue користується шаблоном MVVM для створення інтерфейсів.

Model – view – viewmodel (MVVM) - це програмний архітектурний шаблон, який полегшує відокремлення розвитку графічного інтерфейсу користувача (подання) - будь то за допомогою мови розмітки або коду графічного інтерфейсу - від розвитку бізнес-логіки або зворотного кінцева логіка (модель), щоб подання не залежало від якоїсь конкретної платформи моделі. Модель подання MVVM є перетворювачем значень, тобто модель подання відповідає за викриття (перетворення) об'єктів даних із моделі таким чином, що об'єкти легко управляти та представляти. У цьому відношенні модель подання є більше моделлю, ніж подання, і обробляє більшість, якщо не всю логіку відображення подання. Модель подання може реалізовувати шаблон посередника, організовуючи доступ до внутрішньої логіки навколо набору випадків використання, що підтримуються видом.

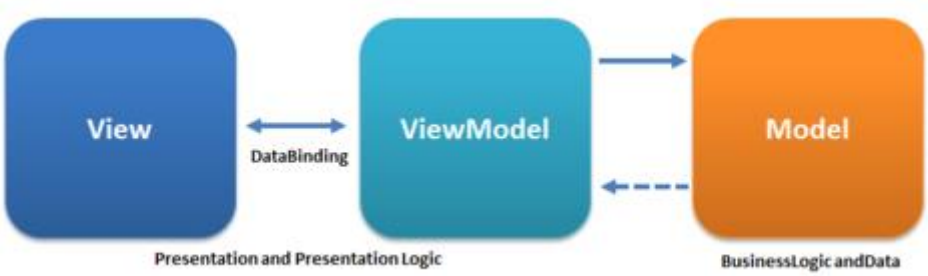


Рис. 1.5 MVVM

Компоненти

Компоненти Vue розширюють основні елементи HTML для інкапсуляції багаторазового коду. На високому рівні компоненти - це власні елементи, до яких компілятор Vue надає поведінку. У Vue компонент, по суті, є екземпляром Vue із заздалегідь визначеними параметрами.

Шаблони

Vue використовує синтаксис шаблону на основі HTML, який дозволяє прив'язувати відтворений DOM до даних базового екземпляра Vue. Усі шаблони Vue є дійсним HTML-кодом, який може бути проаналізований браузерами, що відповідають специфікаціям, і синтаксичними аналізаторами HTML. Vue компілює шаблони у віртуальні функції візуалізації DOM. Віртуальна об'єктна модель документа (або "DOM") дозволяє Vue відображати компоненти в своїй пам'яті перед оновленням браузера. У поєднанні з системою реактивності Vue здатний розрахувати мінімальну кількість компонентів для повторного відтворення та застосувати мінімальну кількість маніпуляцій DOM при зміні стану програми.

Користувачі Vue можуть використовувати синтаксис шаблону або вибрати безпосередньо запис функцій візуалізації за допомогою гіперскрипту або за

допомогою викликів функцій, або JSX. Функції візуалізації дозволяють будувати додаток із програмних компонентів.

Реактивність

Vue сам по собі має систему реактивності, яка використовує прості об'єкти JavaScript та оптимізований рендеринг. Кожен компонент відстежує свої реактивні залежності під час рендерингу, тому система точно знає, коли рендерити і які компоненти рендерити.

Переходи

Vue пропонує різні способи застосування ефектів переходу, коли елементи вставляються, оновлюються або видаляються з DOM. Сюди входять інструменти для:

- Автоматично застосовує класи для переходів CSS та анімації
- Інтегрує сторонні бібліотеки анімації CSS, такі як Animate.css
- Використовує JavaScript для безпосереднього маніпулювання DOM під час перехідних гачків
- Інтегруйте сторонні бібліотеки анімації JavaScript, такі як Velocity.js

Коли елемент, загорнутий у компонент переходу, вставляється або видаляється, відбувається ось що:

- Vue буде автоматично обнюхувати, чи застосовуються в цільовому елементі переходи CSS чи анімація. Якщо це станеться, перехідні класи CSS будуть додані / видалені у відповідні терміни.
- Якщо компонент переходу передбачав хуки JavaScript, ці хуки будуть викликані у відповідні терміни.
- Якщо не виявлено переходів / анімацій CSS і не передбачено переключень JavaScript, операції DOM для вставки та / або видалення будуть виконані негайно на наступному кадрі.

РОЗДІЛ 2

МАРШРУТИЗАЦІЯ. ПІДКЛЮЧЕННЯ ТА НАЛАШТУВАННЯ РОУТЕРІВ ТА ІНШИХ ДОПОМІЖНИХ БІБЛІОТЕК

2.1 Маршрутизація

Традиційним недоліком односторінкових додатків (SPA) є нездатність обміну посиланнями на точну "допоміжну" сторінку в межах певної веб-сторінки. Оскільки SPA обслуговує користувачів лише одну відповідь сервера на основі URL-адреси (зазвичай вона обслуговує index.html або index.vue), закладка певних екранів або обмін посиланнями на певні розділи зазвичай важкі, а то й неможливі. Щоб вирішити цю проблему, багато клієнтських маршрутизаторів розмежовують свої динамічні URL-адреси символом "hashbang" (#), Наприклад page.com/#!/. Однак з HTML5 більшість сучасних браузерів підтримують маршрутизацію без "hashbang" (#).

Vue надає інтерфейс для зміни відображуваного на сторінці на основі поточного шляху до URL-адреси - незалежно від того, як він був змінений (будь-за посиланням електронною поштою, оновленням або посиланнями на сторінці). Крім того, використання інтерфейсного маршрутизатора дозволяє навмисний перехід шляху браузера, коли певні події браузера (наприклад, клацання) відбуваються на кнопках або посиланнях. Сам Vue не постачається з фронт-енд-хеш-маршрутизацією. Але пакет з відкритим вихідним кодом "vue-router" надає API для оновлення URL-адреси програми, підтримує кнопку "Назад" (навігація в історії) і скидання пароля електронної пошти або посилання для перевірки електронної пошти з параметрами URL-адреси автентифікації. Він підтримує відображення вкладених маршрутів до вкладених компонентів і пропонує тонкий контроль переходу. Завдяки Vue розробники вже складають програми з

невеликими будівельними блоками, що створюють більші компоненти. З додаванням маршрутизатора vue до суміші, компоненти повинні бути просто зіставлені з маршрутами, до яких вони належать, а батьківські / кореневі маршрути повинні вказувати, де діти повинні відображатись.

Код з файлу NavMenu.vue:

```
<template>
  <div>
    <div class="navbar">
      <div class="container">
        <div class="navbar-menu">
          <ul>
            <router-link tag="li" to="/">Головна</router-link>
            <router-link tag="li" to="/posts">Публікації</router-link>
            <router-link tag="li" to="/register">Реєстрація</router-link>
            <router-link tag="li" to="/login">Вхід</router-link>
          </ul>
          <p>Вітаю{{isUser}}</p>
        </div>
      </div>
    </div>
    <router-view />
  </div>
</template>

<script>
```

```
export default {
    data() {
        return {
            isUser: ' '
        }
    },
    created() {
        localStorage.getItem('access_token')
        ? this.isUser = `, ${localStorage.getItem('userName') }`
        : this.isUser = ' '
    }
}
</script>

<style lang="scss">
    .container {
        max-width: 1170px;
        margin: auto;
    }

    .navbar {
        background-color: #4caf50;
        overflow: hidden;
        position: fixed;
        top: 0;
        width: 100%;

        &-menu {
```

```
        display: flex;
        justify-content: space-between;

        p {
            font-size: 16px;
            color: #fff;
            padding: 15px 20px;
        }
        ul {
            list-style: none;
            display: flex;
            justify-content: space-between;
            width: 400px;
            li{
                text-transform: uppercase;
                font-family: Roboto, sans-serif;
                font-size: 18px;
                color: #fff;
                padding: 15px 20px;
                cursor: pointer;
            }
        }
    }
  }
</style>
```

Код с файлу router/index.js:

```
import VueRouter from 'vue-router'

import Home from '../pages/Home'
import Posts from '../pages/Posts'
import Register from '../pages/Register'
import Login from '../pages/Login'

export default new VueRouter({
    routes: [
        {
            path: '/',
            name: "Home",
            component: Home
        },
        {
            path: '/posts',
            name: "Posts",
            component: Posts
        },
        {
            path: '/register',
            name: "Register",
            component: Register
        },
        {
            path: '/login',
            name: "Login",
            component: Login
```

```
    }
  ],
  mode: 'history'
})
```

Код вище:

- Встановлює інтерфейсний маршрут за потрібною нам адресою, наприклад websitename.com/login, websitename.com/register та інші.
- Що відображатиметься в компонентах Home, Posts, Login, Register , визначеному в компоненті NavMenu
- Дозволяє користувацькому компоненту передавати конкретну сторінку для користувача, який був введений в URL-адресу за допомогою ключа параметрів об'єкта route: {

  path: '/',

  name: "Home",

  component: Home

  }
- Цей шаблон (залежно від параметрів, переданих у маршрутизатор) буде відображений у <router-link tag="li" to="/"> </router-link> всередині об’єкта DOM#App.
- Остаточно сформований HTML у вигляді списку (ul), за допомогою tag="li" представляє текст маршрутів у вигляді цього тегу "li" .

У файлі router/index.js відбувається налаштування переходів сторінок за допомогою імпортів компонентів у роутери (import Home from '../pages/Home'). Також, це можливо завдяки імпорту самої бібліотеки (import VueRouter from 'vue-router') для роботи з такою технологією.

2.2 Бібліотека axios для взаємодії з REST API

В нашому веб-додатку нам неодноразово знадобитися одержувати і відображати дані з API. Існує кілька способів зробити це, але найбільш популярним рішенням є використання axios, заснованого на Promise HTTP-клієнта.

У цьому випадку ми будемо використовувати REST API для відображення даних, які будуть надсилатись із серверу та самі будемо відсилати ці дані на обробку до сервера. Перш за все, підключимо та налагодимо axios за допомогою npm, yarn або посилання на CDN.

Існує безліч варіантів, як ми можемо запитувати інформацію з API, але спочатку необхідно дізнатися, в якому вигляді надаються дані, щоб розуміти, як їх відображати.

```
import axios from 'axios'

axios.defaults.baseURL = "https://diplommustafaiev.azurewebsites.net/";
```

Тут ми одразу прописали головний шлях до API, щоб не повторювати цю частину коду там де ми будемо постійно звертатись до методів цієї бібліотеки.

Спочатку необхідно створити властивість в data для зберігання нашої інформації, далі винесемо і збережемо дані, використовуючи хук життєвого циклу mounted

```
<script>
import axios from 'axios'

export default {
```

```
data(){
  return {
    Email: "",
    UserName: "",
    Name: "",
    Surname: "",
    MiddleName: "",
    Password: "",
    ConfirmPassword: "",
    showError: false
  }
},
methods: {
  async formSubmit() {
    await axios.post('Api/Account/Register', {
      Email: this.Email,
      UserName: this.UserName,
      Name: this.Name,
      Surname: this.Surname,
      MiddleName: this.MiddleName,
      Password: this.Password,
      ConfirmPassword: this.ConfirmPassword
    })
    .then(() => {
      this.showError = false;
      this.$router.push("Login")
    })
    .catch(() => {
```

```
        this.showError = true;
      })
    }
  }
}
</script>
```

Код вище відсилає данні на обробку до сервера за допомогою методу post.

Метод post може приймаи три параметри, а саме адресу куди ці дані будуть відсилатись ("https://diplommustafaiev.azurewebsites.net/'Api/Account/Register'").

Самі данні, або як в даному випадку об’єкт даних {

```
        Email: this.Email,
        UserName: this.UserName,
        Name: this.Name,
        Surname: this.Surname,
        MiddleName: this.MiddleName,
        Password: this.Password,
        ConfirmPassword: this.ConfirmPassword
      }
```

Та headers – це заголовки, які дозволяють клієнту і серверу відправляти додаткову інформацію з HTTP запитом або відповіддю.

Обробка помилок

Бувають моменти, коли ми не отримали, або неправильно відправили необхідні дані з API. Може бути безліч причин, через які виклик axios міг закінчитися невдачею, наприклад:

- API не був доступний.
- Запит був зроблений неправильно.
- API не надав дані в очікуваному нами форматі.

При виконанні цього запиту ми повинні перевіряти такі обставини і надавати інформацію в кожному випадку, щоб ми знали, як впоратися з цією проблемою. У виклику axios ми зробимо це, використовуючи catch.

```
.catch(() => {
      this.showError = true;
 })
```

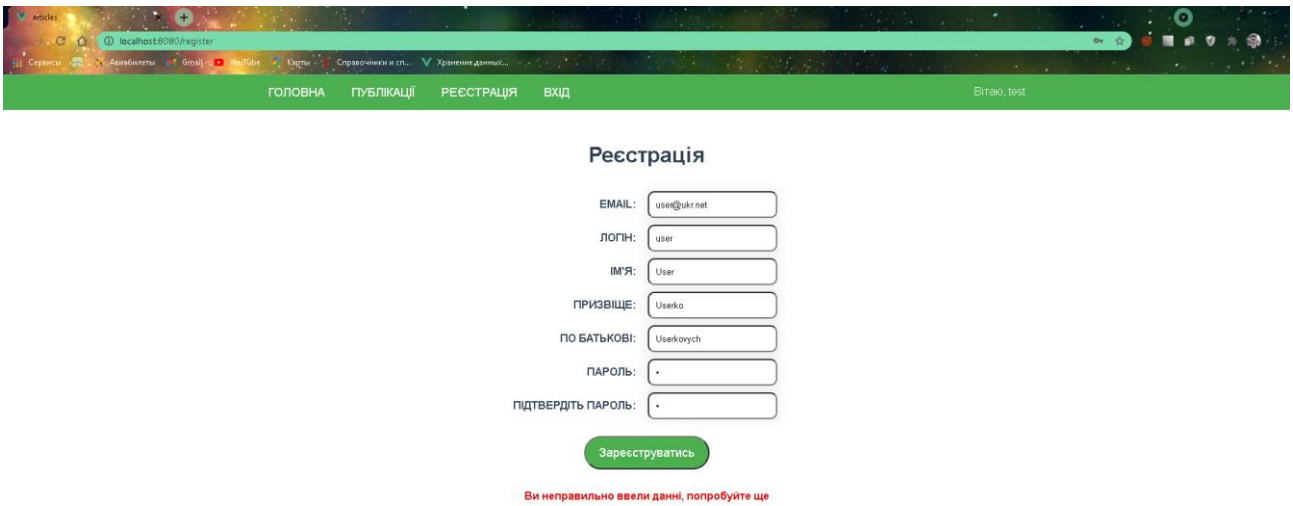


Рис. 2.1 Показ помилки

Відображення даних з API

Типова ситуація, коли необхідна інформація знаходиться всередині відповіді сервера і нам необхідно знати структуру даних для отримання доступу. У

нашому випадку, потрібна інформація публікації знаходиться в resp.data Якщо ми використовуємо це, то запит буде виглядати наступним чином.

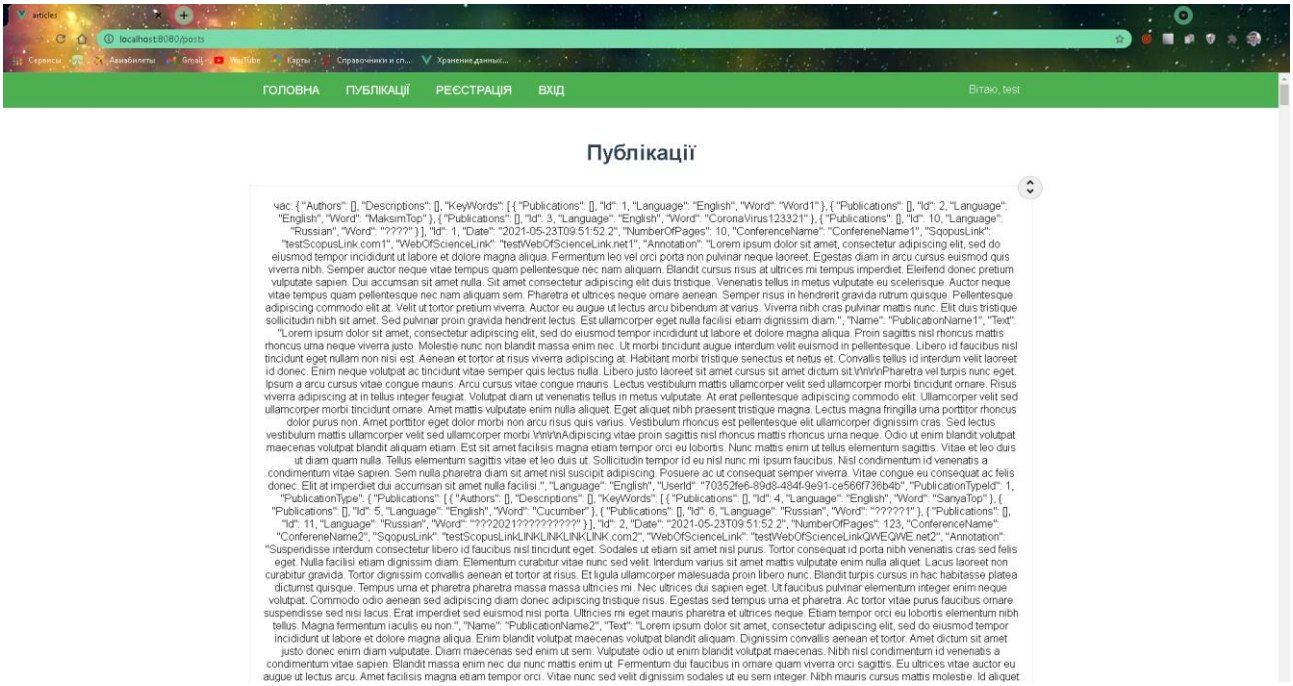


Рис. 2.2 Вивід усіх даних

Завдяки цьому можна працювати з даними, які поступають з сервера та відображати їх так, як нам потрібно.

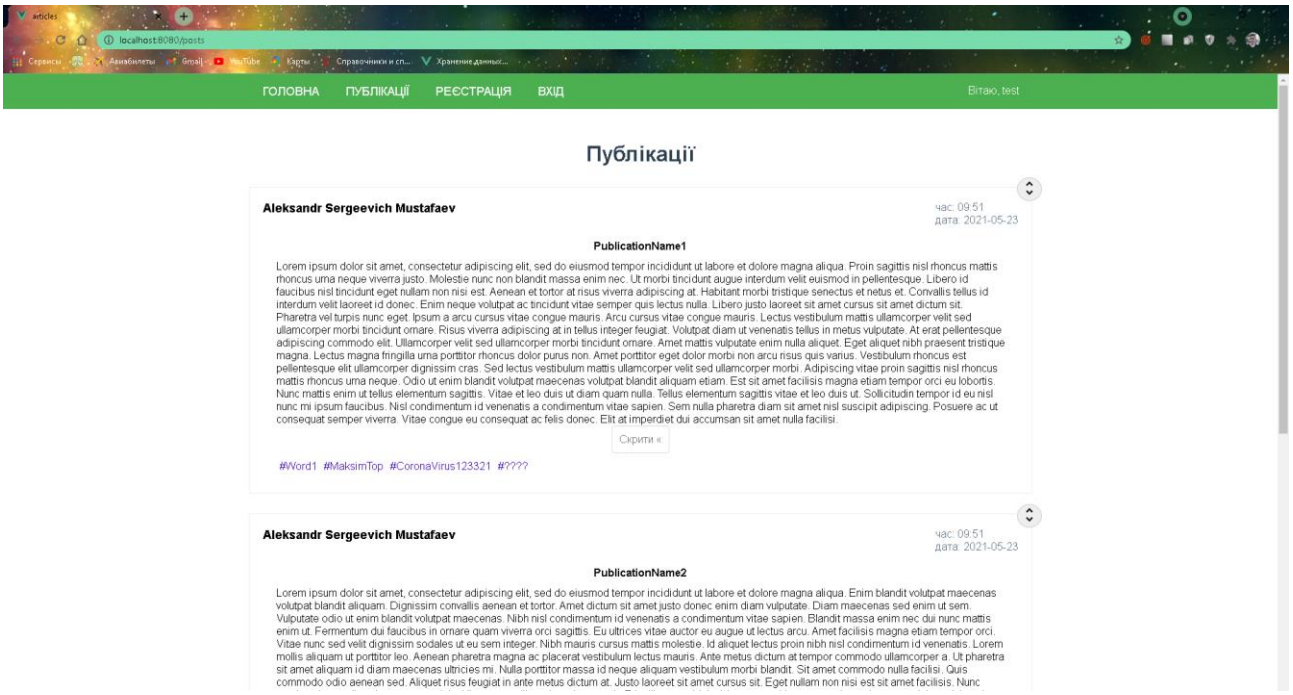


Рис. 2.3 Вивід потрібних даних

РОЗДІЛ 3

ІНТЕРФЕЙС ПРОЕКТУ ТА ЙОГО ФУНКЦІОНАЛ

3.1 Реєстрація користувача

Багато служб соціальних мереж використовують реєстрацію для своїх користувачів задля того, щоб надати їм можливість користуватись цим продуктом та збирати певну інформацію з користувачів.

Щоб користувач мав змогу повністю користуватись веб проектом йому потрібно заповнити форму для реєстрації на сторінці "Реєстрація"

Код з файлу Register.vue:

```
<template>
   <div class="register">
     <h1>Реєстрація</h1>
     <form @submit.prevent="">
       <div>
         <label for="Email">Email:</label>
         <input type="email" v-model="Email" id="Email" />
       </div>
       <div>
         <label for="UserName">Логін:</label>
         <input type="text" v-model="UserName" id="UserName" />
       </div>
       <div>
```

```
        <label for="Name">Ім'я:</label>
        <input type="text" v-model="Name" id="Name" />
      </div>
      <div>
        <label for="Surname">Призвіще:</label>
        <input type="text" v-model="Surname" id="Surname" />
      </div>
      <div>
        <label for="MiddleName">По батькові:</label>
        <input type="text" v-model="MiddleName" id="MiddleName" />
      </div>
      <div>
        <label for="Password">Пароль:</label>
        <input type="password" v-model="Password" id="Password" />
      </div>
      <div>
        <label for="ConfirmPassword">Підтвердіть пароль:</label>
        <input type="password" v-
model="ConfirmPassword" id="ConfirmPassword" />
      </div>
      <button @click="formSubmit" type="submit">Зареєструватись</button>
    </form>
   <p v-
if="showError" id="error">Ви неправильно ввели данні, попробуйте ще</p>
  </div>
</template>

<script>
import axios from 'axios'
```

```
export default {
  data(){
    return {
      Email: "",
      UserName: "",
      Name: "",
      Surname: "",
      MiddleName: "",
      Password: "",
      ConfirmPassword: "",
      showError: false
    }
  },
  methods: {
    async formSubmit() {
      await axios.post('Api/Account/Register', {
        Email: this.Email,
        UserName: this.UserName,
        Name: this.Name,
        Surname: this.Surname,
        MiddleName: this.MiddleName,
        Password: this.Password,
        ConfirmPassword: this.ConfirmPassword
      })
      .then(() => {
        this.showError = false;
        this.$router.push("Login")
```

```
      })
      .catch(() => {
        this.showError = true;
      })
    }
  }
}
</script>

<style lang="scss" scoped>
  * {
    box-sizing: border-box;
  }
  h1 {
    margin: 100px 0 30px 0;
  }
  .register {
    width: 400px;
    margin: 0 auto;
  }
  label {
    padding: 15px 12px 12px 0;
    margin-left: auto;
  }
  form div {
    display: flex;
    justify-content: space-between;
    text-transform: uppercase;
```

```
  font-weight: 700;
}
button[type="submit"] {
  background-color: #4caf50;
  color: white;
  padding: 12px 20px;
  cursor: pointer;
  border-radius: 30px;
  margin-top: 20px;
  font-size: 18px;

  &:hover {
    background-color: #45a049;
  }
}
input {
  margin: 5px;
  box-shadow: 0 0 15px 4px rgba(0, 0, 0, 0.06);
  padding: 10px;
  border-radius: 10px;

  &:focus {
    outline: none;
  }
}
#error {
  color: rgb(230, 1, 1);
  margin-top: 30px;
```

```
    font-weight: 600;
  }
</style>
```

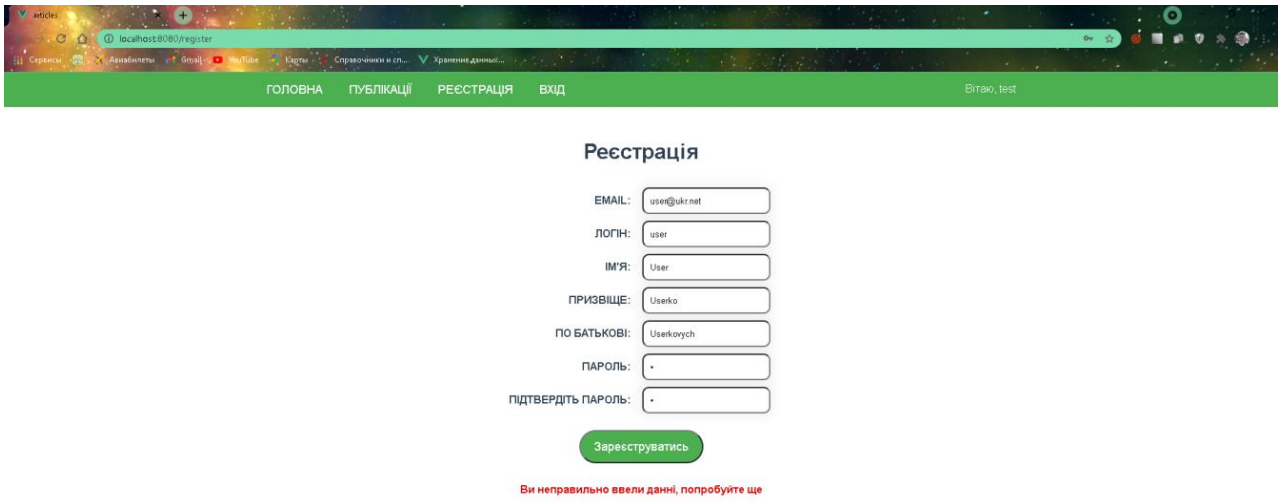


Рис. 3.1 Форма реєстрації

Дані, які увів користувач містяться в об’єкті який відсилається на обробку до сервера.

Якщо користувач неправильно заповнив свої дані то сервер поверне помилку, яку користувач зможе побачити та виправити.

Всі данні обов’язково мають бути заповненими.

В успішному випадку користувача перекидує на сторінку входу.

3.2 Вхід користувача

Після того як користувач успішно пройшов етап реєстрування йому потрібно здійснити вхід у систему, щоб повноцінно нею користуватися.

Код з файлу Login.vue:

```
<template>
   <div class="login">
      <h1>Вхід</h1>
      <form @submit.prevent="logIn">
      <div>
        <label for="userName">Логін:</label>
        <input type="text" v-model="userName" id="userName" />
      </div>
      <div>
        <label for="Password">Пароль:</label>
        <input type="password" v-model="Password" id="Password" />
      </div>
      <button type="submit" >Вхід</button>
     </form>
     <p v-
if="showError" id="error">Ви неправильно ввели данні, попробуйте ще раз</p>
   </div>
</template>

<script>
```

```
export default {
  data() {
     return {
       userName: "",
       Password: "",
       showError: false
     }
  },
  methods: {
    async logIn() {
     const axios = require('axios')

     const params = new URLSearchParams()
     params.append( 'userName', this.userName)
     params.append( 'Password', this.Password)
     params.append('grant_type', 'password')

     const config = {
       headers: {
         'Content-Type': 'application/x-www-form-urlencoded'
       }
     }

     await axios.post('token', params, config)
     .then(res => {
       localStorage.setItem('access_token' , res.data.access_token);
       localStorage.setItem('userName' , res.data.userName);
       this.showError = false;
```

```
        this.$router.push("Posts")
        window.location.reload();
      })
      .catch(() => {
        this.showError = true;
      })
    }
  }
}
</script>

<style lang="scss" scoped>
  * {
    box-sizing: border-box;
  }
  h1 {
    margin: 100px 0 30px 0;
  }
  .login {
    width: 400px;
    margin: 0 auto;
  }
  label {
    padding: 15px 12px 12px 0;
    margin-left: auto;
  }
  form div {
    display: flex;
```

```
  justify-content: space-between;
  text-transform: uppercase;
  font-weight: 700;
}
button {
  background-color: #4caf50;
  color: white;
  padding: 12px 20px;
  cursor: pointer;
  border-radius: 30px;
  margin-top: 20px;
  font-size: 18px;

  &:hover {
    background-color: #45a049;
  }
}
input {
  margin: 5px;
  box-shadow: 0 0 15px 4px rgba(0, 0, 0, 0.06);
  padding: 10px;
  border-radius: 10px;

  &:focus {
    outline: none;
  }
}
#error {
```

```
    color: rgb(230, 1, 1);
    margin-top: 30px;
    font-weight: 600;
  }
</style>
```

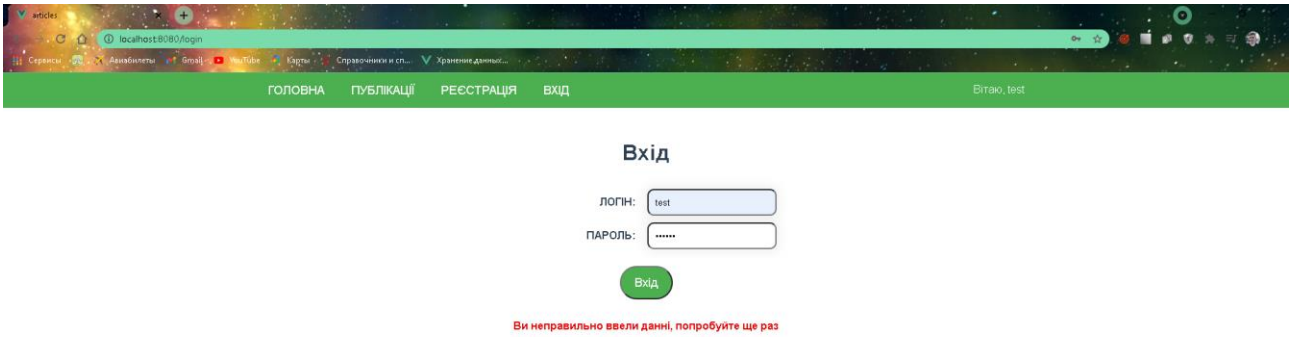


Рис. 3.2 Вхід у систему

У разі неправильного вводу даних виводиться помилка для користувача.

В асинхронному методі logIn() задаються параметри, які мають надіслатися для підтвердження сервером, шо такий користувач дійсно зареєстрований.

Якщо користувач успішно проходить етап входу у систему йому надається JWT токен, який буде зберігатись в local storage.

Відкритий стандарт JWT офіційно з'явився в 2015 обіцяючи цікаві особливості і широкі перспективи. Правильне зберігання Access токена є життєво важливим

питанням при побудові системи авторизації та аутентифікації в сучасному Web, де стають дедалі популярнішими сайти, побудовані за технологією SPA.

Неправильне зберігання токенів веде до їх крадіжці і перевикористанням зловмисниками.

Токен зберігається у цій змінній res.data.access_token після успішного виконання запросу.

Коли користувач отримує даний токен йому надається змога користуватись веб застосунком у повній мірі.

ВИСНОВОК

Данна робота була написана з використанням сучасних технологій, які надають великі можливості у розробці та проектуванні даного продукту. Завдяки цьому створений програмний продукт був детально пророблений за всіма основними концепціями програмування.

Виконаний проект демонструє перспективність використання компонентного підходу та правила написання чистого коду. Була розроблена клієнтська частина веб застосунку для публікації звітів.

Спроектована структура проекту дозволяє й надалі розширювати функціонал нашого застосунку, оскільки при написанні були застосовані основні концепції проектування проекту.

Це програмне забезпечення дозволить створювати публікації та дивитись публікації які створили інші користувачі додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. JavaScript. Детальний керівництво. 6-е видання, Девід Фленаган. – 2017. – С. 232 – 247.

2. JavaScript: сильні сторони. Дуглас Крокфорд. – 2012. – С. 100 – 116.

3. Офіційна документація Vue – https://vuejs.org/v2/guide/index.html.

4. Грокаєм алгоритми. Ілюстрований посібник для програмістів і цікавих – 2021.

Додаток А

ВИХІДНИЙ КОД ПРОГРАМИ

NavMenu.vue

```
<template>
  <div>
    <div class="navbar">
      <div class="container">
        <div class="navbar-menu">
          <ul>
            <router-link tag="li" to="/">Головна</router-link>
            <router-link tag="li" to="/posts">Публікації</router-link>
            <router-link tag="li" to="/register">Реєстрація</router-link>
            <router-link tag="li" to="/login">Вхід</router-link>
          </ul>
          <p>Вітаю{{isUser}}</p>
        </div>
      </div>
    </div>
    <router-view />
  </div>
</template>

<script>

export default {
  data() {
```

```
        return {
            isUser: ''
        }
    },
    created() {
        localStorage.getItem('access_token')
        ? this.isUser = `, ${localStorage.getItem('userName') }`
        : this.isUser = ''
    }
}
</script>

<style lang="scss">
    .container {
        max-width: 1170px;
        margin: auto;
    }

    .navbar {
        background-color: #4caf50;
        overflow: hidden;
        position: fixed;
        top: 0;
        width: 100%;
        z-index: 4;

        &-menu {
            display: flex;
```

```
        justify-content: space-between;

        p {
            font-size: 16px;
            color: #fff;
            padding: 15px 20px;
        }
        ul {
            list-style: none;
            display: flex;
            justify-content: space-between;
            width: 400px;
            li{
                text-transform: uppercase;
                font-family: Roboto, sans-serif;
                font-size: 18px;
                color: #fff;
                padding: 15px 20px;
                cursor: pointer;
            }
        }
    }
  }
</style>
```

Posts.vue

```
<template>
  <div>
    <div class="container">
      <h1>Публікації</h1>
      <div v-if="loading" class="lds-
roller"><div></div><div></div><div></div><div></div><div></div><div></div><
div></div><div></div></div>

      <div v-for="item in posts" :key="item.id">
        <article class="blog-card">

          <button class="show-modal" id="show-
modal" @click="showModal = true">
            <svg version="1.1" id="Capa_1" xmlns="http://www.w3.org/2000/sv
g" xmlns:xlink="http://www.w3.org/1999/xlink" x="0px" y="0px"
              width="35px" height="20px" viewBox="0 0 459 459" style="enabl
e-background:new 0 0 459 459;" xml:space="preserve">
              <g id="unfold-more">
                <path d="M229.5,71.4l81.6,81.6l35.7-
35.7L229.5,0L112.2,117.3l35.7,35.7L229.5,71.4z M229.5,387.6L147.9,306l-
35.7,35.7L229.5,459
                  l117.3-117.3L311.1,306L229.5,387.6z"/>
              </g>
            </svg>
          </button>
          <transition name="fade" appear>
            <div
              class="modal-ovarlay"
              v-if="showModal"
              @click="showModal = false">
```

```
            </div>
        </transition>
        <transition name="slide" appear>
            <div class="modal" v-if="showModal" >
                <h2>Додаткова інформація</h2>
                <p>Журнал: {{item.Publication.Journal.Name}}</p>
                <p>Мова: {{item.Publication.Language}}</p>
                <p>Посилання на статю web of science: <a href="#">{{item.Publi
cation.WebOfScienceLink}}</a></p>
                <p>Посилання на статю sqopus: <a href="#">{{item.Publication.
SqopusLink}}</a></p>
                <p>Конференція: {{item.Publication.Conference.Name}}</p>
                <p>Кількість сторінок: {{item.Publication.NumberOfPages}}</p>
                <button class="hide-
modal" @click="showModal = false">Закрити</button>
            </div>
        </transition>

        <div class="blog-card__head">
            <h4 class="user">{{`${item.User.Name} ${item.User.MiddleName}
${item.User.Surname}`}}</h4>
            <div class="time">
                <p>час: {{item.Publication.Date.slice(11,-5)}}</p>
                <p>дата: {{item.Publication.Date.slice(0, -11)}}</p>
            </div>
        </div>
        <div class="blog-card__info">
            <h5 class="title">{{item.Publication.Name}}</h5>
```

```
                    <input type="checkbox" class="show-text" id="show-text" />
                    <div class="limited l-300">
                        <p class="text">{{item.Publication.Text}}</p>
                        <div class="bottom"></div>
                    </div>
                     <label for="show-text" class="show-text-btn"></label>
                    <ul class="nav-keyword">
                        <li
                            v-for="keyItem in item.Publication.KeyWords"
                            :key="keyItem.Id"
                        >#{{keyItem.Word}}</li>
                    </ul>
                </div>
            </article>
        </div>
    </div>
</template>

<script>
import axios from 'axios'

export default {
    data() {
        return {
            posts: null,
            loading: true,
            showModal: false
```

```
        }
    },
    async mounted() {
        await axios.get('Api/Publication/Get/', {
            headers: {
                "Authorization" : `Bearer ${localStorage.getItem('access_token')}`
            }
        })
        .then(res => (this.posts = res.data))
        .catch(() => {
            this.$router.push("Login")
        })
        .finally(() => {
            this.loading = false
        })
        console.log(this.posts[0].User)
    }
}
</script>

<style lang="scss" scoped>
    * {
        box-sizing: border-box;
    }
    h1 {
        margin: 100px 0 30px 0;
    }
```

```
.blog-card {
    position: relative;
    border: 1px solid rgb(235, 238, 240);
    box-sizing: border-box;
    padding: 20px;
    color: #000;
    margin-bottom: 30px;

    &__head {
        display: flex;
        justify-content: space-between;

        .user {
          font-size: 18px;
        }

        .time {
            color: rgb(91, 112, 131);
            text-align: left;
        }
    }

    &__info{
        padding: 0 20px;

        .title {
            font-size: 16px;
            margin: 20px 0 10px;
```

```
        }
        .text {
            text-align: left;
        }
    }
}

.nav-keyword {
    display: flex;
    list-style: none;
    li {
        color: rgb(57, 4, 182);
        margin: 10px 5px;
        cursor: pointer;

        &:hover {
            color: darken(rgb(57, 4, 182), 15%);
        }
    }
}

// модальне вікно

.show-modal {
    position: absolute;
    top: -15px;
    right: -15px;
    border-radius: 50%;
```

```
    border: 1px solid #ccc;
    padding: 5px 0;
    cursor: pointer;
}
.hide-modal {
    cursor: pointer;
    padding: 10px;
    color: #777;
    border: 1px solid #ddd;
    background: #fff;
    border-radius: 4px;
}

.modal-ovarlay {
    position: absolute;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    z-index: 1;
    background-color: rgba(0, 0, 0, 0.3);
}

.modal {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
```

```
    z-index: 2;
    text-align-last: left;
    padding: 20px;

    width: 100%;
    max-width: 800px;
    background-color: #fff;

    p {
        margin: 10px 0;
    }
}

.fade-enter-active,
.fade-leave-active {
    transition: opacity 0.5s;
}

.fade-enter,
.fade-leave-to {
    opacity: 0;
}

// Анімація загрузки

.lds-roller {
    display: inline-block;
    position: relative;
```

```
    width: 80px;
    height: 80px;
}
.lds-roller div {
    animation: lds-roller 1.2s cubic-bezier(0.5, 0, 0.5, 1) infinite;
    transform-origin: 40px 40px;
}
.lds-roller div:after {
    content: " ";
    display: block;
    position: absolute;
    width: 7px;
    height: 7px;
    border-radius: 50%;
    background: #4caf50;
    margin: -4px 0 0 -4px;
}
.lds-roller div:nth-child(1) {
    animation-delay: -0.036s;
}
.lds-roller div:nth-child(1):after {
    top: 63px;
    left: 63px;
}
.lds-roller div:nth-child(2) {
    animation-delay: -0.072s;
}
.lds-roller div:nth-child(2):after {
```

```
    top: 68px;
    left: 56px;
}
.lds-roller div:nth-child(3) {
    animation-delay: -0.108s;
}
.lds-roller div:nth-child(3):after {
    top: 71px;
    left: 48px;
}
.lds-roller div:nth-child(4) {
    animation-delay: -0.144s;
}
.lds-roller div:nth-child(4):after {
    top: 72px;
    left: 40px;
}
.lds-roller div:nth-child(5) {
    animation-delay: -0.18s;
}
.lds-roller div:nth-child(5):after {
    top: 71px;
    left: 32px;
}
.lds-roller div:nth-child(6) {
    animation-delay: -0.216s;
}
.lds-roller div:nth-child(6):after {
```

```
    top: 68px;
    left: 24px;
}
.lds-roller div:nth-child(7) {
    animation-delay: -0.252s;
}
.lds-roller div:nth-child(7):after {
    top: 63px;
    left: 17px;
}
.lds-roller div:nth-child(8) {
    animation-delay: -0.288s;
}
.lds-roller div:nth-child(8):after {
    top: 56px;
    left: 12px;
}
@keyframes lds-roller {
    0% {
        transform: rotate(0deg);
    }
    100% {
        transform: rotate(360deg);
    }
}

// скрити/показати текст
```

```
.limited {
    max-height: 400px;
    overflow: hidden;
    position: relative;
}

.limited.l-300 {
    max-height: 300px;
}
.limited .bottom {
    position: absolute;
    bottom: 0;
    background: linear-
gradient(to bottom, rgba(255,255,255,0), rgba(255,255,255,1) 80%);
    width: 100%;
    height: 60px;
    opacity: 1;
    transition: .3s;
}
.show-text {
    opacity: 0;
    position: absolute;
}
.show-text:checked ~ .limited {
    max-height: none;
}
.show-text:checked ~ .limited .bottom {
    opacity: 0;
```

```
        transition: .3s;
    }
    .show-text ~ .show-text-btn:before {
        content: 'Показати »';
    }
    .show-text:checked ~ .show-text-btn:before {
        content: 'Скрити «';
    }
    .show-text-btn {
        cursor: pointer;
        display: inline-block;
        padding: 10px;
        color: #777;
        border: 1px solid #ddd;
        border-radius: 4px;
    }
</style>
```

Додаток Л

Taras Shevchenko National University of Kyiv

Design and development of the client part of the web application for maintaining a database of publications

Software Architecture Document (SAD)

Content Owner: Maksym Stoliar

DOCUMENT NUMBER: 1.0

RELEASE: 1.0

RELEASE DATE: 01.06.2021

TABLE OF CONTENTS

1. Introduction

    1.1. Purpose

    1.2. Scope

    1.3. Definitions

    1.4. References

2. Architecture Representation

3. Architectural Goals and Constraints

4. Use Case diagram

5. Size and Performance

1. Introduction

1.1. Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

1.2. Scope

This Software Architecture Document provides an architectural overview of the SPA. SPA has its own multi task list.

1.3. Definitions

SPA - single-page application, which serves as the basic logic between the interaction of the client and the database. 1.4. References

Applicable references are:

- SPA
- REST API
- Vue.js
- DOM

2. **Architecture Representation**

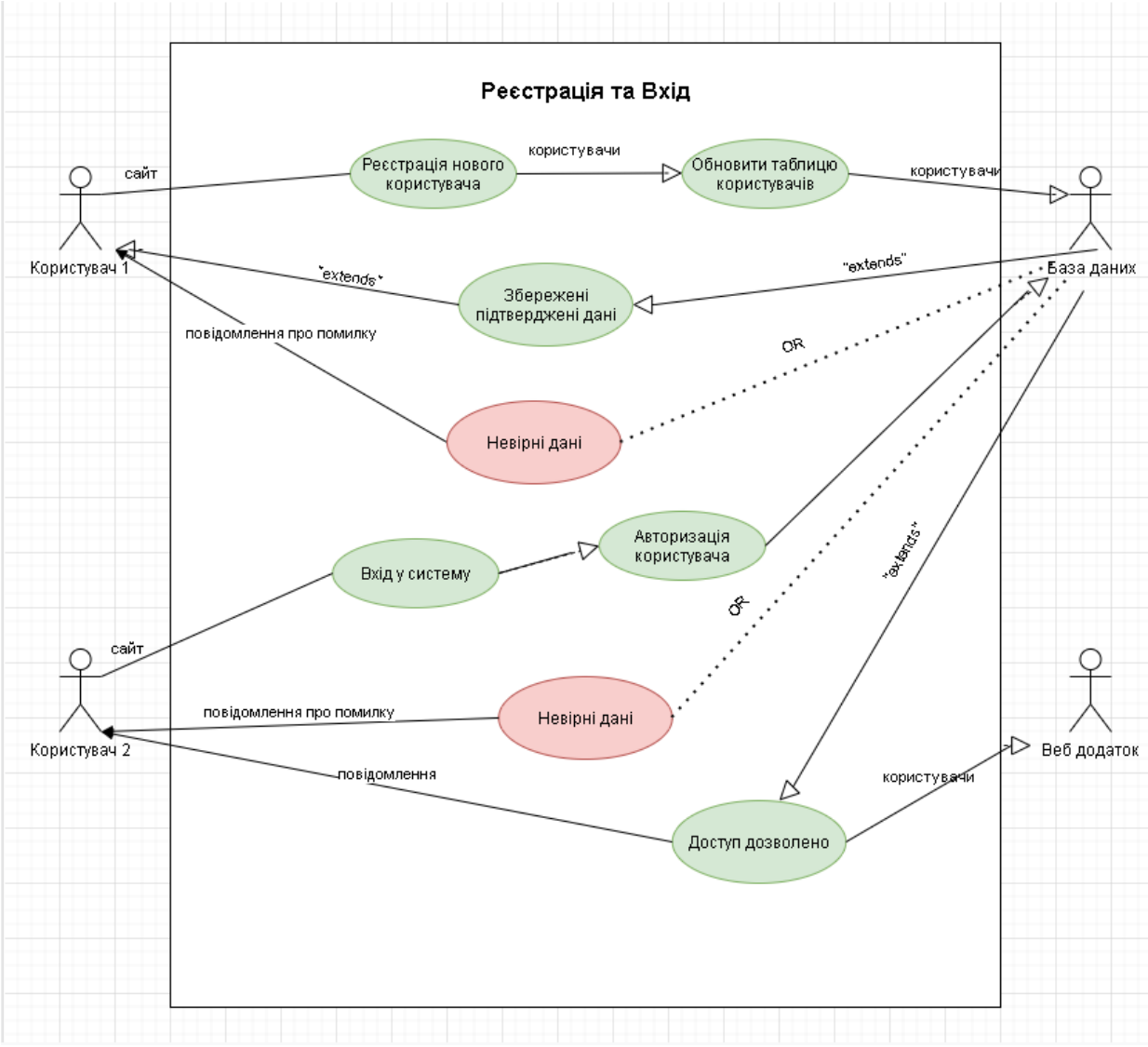
Thanks to routers, which allow you to divide all the logic of the program into four parts and simplify the program and its structure. Also, the axios library was used to communicate with the database, which allows you to conveniently send and download data from the server.

3. **Architectural Goals and Constraints**

- The user cannot use the system until registration and login.
- The user can make changes only in their publications.

4. **Use Case diagram**

This Use Case below illustrates the User registering for the first time within the application and the response given. It also illustrates the exception, if the details are not correct. This Use case also illustrates the Login process after registration and the exception, if details are incorrect.



5. **Size and Performance**

The chosen software architecture supports multithreading and is well optimized for many users.

Requirements:

• The chosen software architecture supports multithreading and is well optimized for many users.

• The system must support up to 10,000 concurrent users.

• The system provides access to the information of the dataset in a few seconds.

• The system must be able to fulfill any design requirements.